Exact algorithms for computing the tree edit distance between unordered trees

Akutsu, Tatsuya; Fukagawa, Daiji; Takasu, Atsuhiro; Tamura, Takeyuki

Exact Algorithms for Computing Tree Edit Distance between Unordered Trees

Tatsuya Akutsu[1]*, Daiji Fukagawa[2], Atsuhiro Takasu[2], Takeyuki Tamura[1]

[1] Bioinformatics Center, Institute for Chemical Research, Kyoto University, Gokasho, Uji, Kyoto, 611-0011, Japan.

[2] National Institute of Informatics, Tokyo 101-8430, Japan.

Abstract

This paper presents a fixed-parameter algorithm for the tree edit distance problem for unordered trees under the unit cost model that works in $O(2.62^k \cdot poly(n))$ time and $O(n^2)$ space, where the parameter k is the maximum bound of the edit distance and n is the maximum size of input trees. This paper also presents polynomial time algorithms for the case where the maximum degree of the largest common subtree is bounded by a constant.

Keywords: Tree edit distance, Fixed parameter algorithms, Dynamic programming, Unordered trees

1 Introduction

Tree pattern matching is one of the well-studied pattern matching problems on graphs. It is important for such application areas as computational biology, XML databases and image analysis [1, 12]. Though various measures have been proposed for computing the similarity between trees, the *edit distance* between rooted trees has been well-studied.

For the tree edit distance problem for ordered trees, Tai developed an $O(n^6)$ time algorithm [13], from which several improvements followed. Demaine *et al.* developed an $O(n^3)$ time algorithm and showed that this bound is optimal under some computation strategy [3].

However, in some applications, it is preferable to regard input trees as unordered trees. At least, in many applications, more flexible matching can be made possible if input trees are regarded as unordered trees. Unfortunately, Zhang *et al.* proved that the tree edit distance problem for unordered trees is NP-hard [14]. Furthermore, Zhang and Jiang proved that it is MAX SNP-hard [15], which means that there exists no polynomial time approximation scheme unless P=NP.

Some optimal algorithms were proposed for restricted cases [8, 12, 16]. Shasha *et al.* developed a fixed-parameter algorithm when the parameter is the number of leaves [12]. Halldórsson and Tanaka developed a polynomial time algorithm for the case of bounded number of branching nodes [8]. Zhang developed a polynomial time algorithm under restricted editing operations [16]. For a related tree inclusion problem, Kilpeläinen and Mannila showed that it is solved in polynomial time if the maximum degree is bounded by a constant

*Corresponding author

[11]. However, the above cases do not cover all important cases and thus polynomial time algorithms should be developed for other cases.

In this paper, we present a fixed-parameter algorithm that works in $O(2.62^k \cdot poly(n))$ time in $O(n^2)$ space, where the parameter k is the edit distance and n is the maximum size of input trees. This algorithm is based on the idea in a fixed-parameter algorithm for computing the maximum common subtrees (in the sense of usual subgraphs, not in the sense of this paper) previously developed by some of the authors [7] [1]. However, it is far from a simple extension of [7] because novel ideas are introduced in the algorithm and analysis presented here. We also present polynomial time algorithms for the case in which the maximum degree of the largest common subtree (based on editing operations) is bounded by a constant. This result is interesting since the tree edit distance problem remains NP-hard even if the maximum degree of input trees is bounded [14].

2 Preliminaries

We briefly review the definitions of tree edit distance, edit distance mapping and largest common subtree (see also Fig. 1) for rooted, labeled and unordered trees [1, 14, 15].

Let T be a rooted unordered tree, where each node v has a label $\ell(v)$ over an alphabet Σ. $r(T)$ and $V(T)$ denote the root of T and the set of nodes in T, respectively. For a node $v \in V(T)$, $T - v$ denotes the tree obtained by deleting v from T, $p(v)$ denotes the parent of v, $chd(v)$ denotes the set of children of v, $deg(v)$ denotes the outdegree of v, $anc(v)$ denotes the set of ancestors of v, $des(v)$ denotes the set of descendants of v. It should be noted that $deg(v) = |chd(v)|$ holds.

$T(v)$ denotes the subtree induced by v and its descendants. For a subtree T' of T, $T - T'$ denotes the tree obtained by deleting all nodes in T' from T. The *depth* of a node v is the length of the path from the root to v and is denoted by $depth(v)$. For a set of nodes $\{v_1, v_2, \ldots, v_h\} \subseteq V(T)$, $LCA(\{v_1, v_2, \ldots, v_h\})$ denotes the lowest common ancestor of $v_1, v_2, \ldots, v_h$. That is, $LCA(\{v_1, v_2, \ldots, v_h\})$ is the deepest node v such that v is an ancestor of each v_i $(i = 1, \ldots, h)$. If T_1 and T_2 are isomorphic including label information, we write $T_1 \approx T_2$. In the analysis of algorithms, n denotes $\max\{|V(T_1)|, |V(T_2)|\}$.

An *edit operation on a tree* T is either a deletion, an insertion, or a substitution, where each operation is defined as follows (see also Fig. 1):

Deletion: Delete a non-root node v in T with parent u, making the children of v become children of u. The children are inserted in the place of v into the set of the children of u.

Insertion: Inverse of delete. Insert a node v as a child of u in T, making v the parent of some of the children of u.

Substitution: Change the label of a node v in T.

We assign a *cost* for each editing operation: $\gamma(a, b)$ denotes the cost of substituting a node with label a to label b, $\gamma(a, \epsilon)$ denotes the cost of deleting a node labeled with a, and $\gamma(\epsilon, a)$ denotes the cost of inserting a node labeled with a.

The *edit distance* between two unordered trees T_1 and T_2 is defined as the cost of the minimum cost sequence of editing operations that transforms T_1 to T_2. We use $dist(T_1, T_2)$

[1] We found that the results in Section 5.2 of [7] were incorrect because we incorrectly assumed that tree alignment satisfied the conditions on a distance metric.

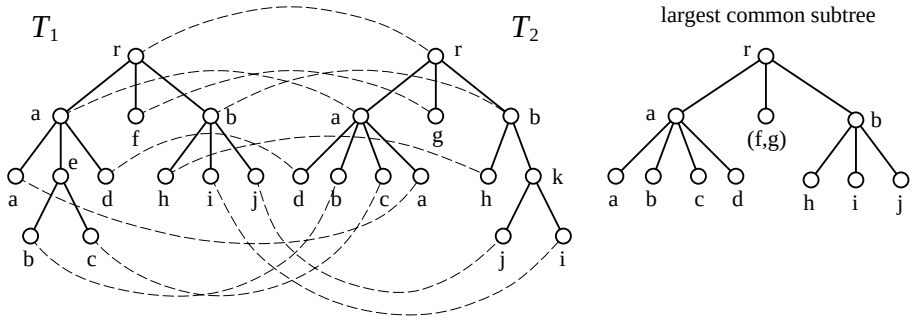

Figure 1: Example of tree edit operation, mapping, and largest common subtree under the unit cost model. T_2 is obtained from T_1 by deletion of node (labeled with) e, insertion of node k and substitution of node f. The corresponding mapping M is shown by broken curves. The largest common subtree is shown in the right-hand side, where the labels of the original nodes are shown in place of the original node pairs. The node labeled (f,g) is not included in the usual largest common subtree [15].

to denote the edit distance between T_1 and T_2. In this paper, we adopt the following standard assumption so that $dist(T_1, T_2)$ becomes a distance metric [1, 14]: $\gamma(a, b) \geq 0$ for any $(a, b) \in \Sigma' \times \Sigma'$, $\gamma(a, a) = 0$ for any $a \in \Sigma'$, $\gamma(a, b) = \gamma(b, a)$ for any $(a, b) \in \Sigma' \times \Sigma'$, $\gamma(a, c) \leq \gamma(a, b) + \gamma(b, c)$ for any $a, b, c \in \Sigma' \times \Sigma' \times \Sigma'$, where $\Sigma' = \Sigma \cup \{\epsilon\}$. We call T_2 a *subtree* of T_1 if T_2 is obtained from T_1 only by deletion operations[2].

It is known that there exists a close relationship between the edit distance and the *edit distance mapping* (or just *mapping*) [1, 14]. $M \subseteq V(T_1) \times V(T_2)$ is called a mapping if the following conditions are satisfied for any two pairs $(v_1, w_1), (v_2, w_2) \in M$: $v_1 = v_2$ iff $w_1 = w_2$, v_1 is an ancestor of v_2 iff w_1 is an ancestor of w_2. Let I_1 and I_2 be the sets of nodes in $V(T_1)$ and $V(T_2)$ not appearing in M, respectively. Then, it is known [1, 14] that the following relation holds:

$$dist(T_1, T_2) = \min_M \left\{ \sum_{u \in I_1} \gamma(\ell(u), \epsilon) + \sum_{v \in I_2} \gamma(\epsilon, \ell(v)) + \sum_{(u,v) \in M} \gamma(\ell(u), \ell(v)) \right\}.$$

In order to relate the edit distance mapping to the largest common subtree, we define a *score function*, which gives a kind of similarity measure between two nodes. Different from a cost function, the value of a score function is higher if two nodes are similar to each other. For each $(u, v) \in V(T_1) \times V(T_2)$, we define a score function $f(u, v)$ by $f(u, v) = \gamma(\ell(u), \epsilon) + \gamma(\epsilon, \ell(v)) - \gamma(\ell(u), \ell(v))$. It is seen that $f(u, v) = f(v, u) \geq 0$ holds. It should also be noted that under the unit cost model (i.e., $\gamma(a, b) = 1$ for all $\ell(a) \neq \ell(b)$), $f(v, v) = 2$ and $f(u, v) = 1$ hold for $\ell(u) \neq \ell(v)$. Let $score(M)$ be the score of a mapping M defined by $score(M) = \sum_{(u,v) \in M} f(u, v)$. Let M_{OPT} be a mapping with the maximum score. Then, we can see from the definition that the following equality holds, where we assume w.l.o.g. that

[2]We also use the *subtree* for denoting a subgraph of a tree. However, the meaning of the subtree is clear from the context and thus there is no confusion.

the roots of T_1 and T_2 correspond to each other in M_{OPT}:

$$\begin{aligned}
dist(T_1,T_2) &= \min_M \left\{ \sum_{u\in I_1} \gamma(\ell(u),\epsilon) + \sum_{v\in I_2} \gamma(\epsilon,\ell(v)) + \sum_{(u,v)\in M} \gamma(\ell(u),\ell(v)) \right\} \\
&= \min_M \left\{ \sum_{u\in V(T_1)} \gamma(\ell(u),\epsilon) + \sum_{v\in V(T_2)} \gamma(\epsilon,\ell(v)) \right. \\
&\qquad \left. + \sum_{(u,v)\in M} (\gamma(\ell(u),\ell(v)) - \gamma(\ell(u),\epsilon) - \gamma(\epsilon,\ell(v))) \right\} \\
&= \sum_{u\in V(T_1)} \gamma(\ell(u),\epsilon) + \sum_{v\in V(T_2)} \gamma(\epsilon,\ell(v)) \\
&\qquad - \max_M \left\{ \sum_{(u,v)\in M} (\gamma(\ell(u),\epsilon) + \gamma(\epsilon,\ell(v)) - \gamma(\ell(u),\ell(v))) \right\} \\
&= \sum_{u\in V(T_1)} \gamma(\ell(u),\epsilon) + \sum_{v\in V(T_2)} \gamma(\epsilon,\ell(v)) - score(M_{OPT}).
\end{aligned}$$

It is to be noted that the first and second terms in the right hand side of the last equality are invariant with a mapping. Therefore, this equality means that the edit distance can be obtained by computing a mapping with the maximum score.

If M consists of pairs of identical labels, the subtree obtained by deleting nodes not appearing in M from T_1 is isomorphic to the subtree obtained by deleting nodes not appearing in M from T_2. Such a tree is called a *common subtree* between T_1 and T_2. In this paper, a subtree of T_1 (or T_2) induced by the nodes appearing in M is also called a common subtree even if M contains some pairs of non-identical labels. The *largest common subtree* (LCST, in short) is defined as the common subtree with the maximum score.

Though the edit distance problem for unordered trees is NP-hard, it can be solved (in exponential time) using a dynamic programming (DP) algorithm [1, 8]. For a forest (i.e., a set of unordered trees) F, $roots(F)$ denotes a set of the roots of trees in F. We define $\delta(F_1,F_2)$ between two unordered forests F_1 and F_2 by the following DP procedure:[3]

$$\begin{aligned}
\delta(F_1,\epsilon) &= \sum_{u\in V(F_1)} \gamma(\ell(u),\epsilon), \\
\delta(\epsilon,F_2) &= \sum_{v\in V(F_2)} \gamma(\epsilon,\ell(v)), \\
\delta(F_1,F_2) &= \min \left\{ \begin{array}{l} \min_{u\in roots(F_1)} \{\delta(F_1-u,F_2) + \gamma(\ell(u),\epsilon)\}, \\ \min_{v\in roots(F_2)} \{\delta(F_1,F_2-v) + \gamma(\epsilon,\ell(v))\}, \\ \min_{(u,v)\in roots(F_1)\times roots(F_2)} \{\delta(F_1-T_1(u),F_2-T_2(v)) \\ \qquad +\delta(T_1(u)-u,T_2(v)-v) + \gamma(\ell(u),\ell(v))\}. \end{array} \right.
\end{aligned}$$

Then, it is seen that $dist(T_1,T_2) = \delta(T_1,T_2)$ holds from [1, 8].

[3]The roots need not correspond to each other in this procedure. However, we can let roots correspond to each other by setting deletion costs for the roots very large.

3 Fixed-Parameter Algorithm

In this section, we present an $O(2.62^k \cdot poly(n))$ time algorithm for the tree edit distance problem between two unordered trees, where the parameter is the edit distance. For details of fixed-parameter algorithms, refer to [5, 6]. Though we present the algorithm for the unit cost model for the simplicity, we will show that the algorithm can be extended for a more general case in which costs of edit operations are integers and $\gamma(a, b) > 0$ holds for all $a \neq b$.

The algorithm is based on the DP algorithm presented in Section 2 and a fixed parameter algorithm for computing the maximum common subtrees (in the sense of usual subgraphs) [7]. However, it is far from a simple extension of combination of these algorithms.

First, we present important lemmas for developing the fixed parameter algorithm.

Lemma 1 *(See also Fig. 2) Let r_1 and r_2 be the roots of T_1 and T_2 respectively. Suppose that for any pair $(u, v) \in chd(r_1) \times chd(r_2)$, $dist(T_1(u), T_2(v)) \geq 1$ holds. Suppose that u is an arbitrary child of r_1 or r_2 such that $|T_1(u)|$ is the largest, where we assume w.l.o.g. that $u \in chd(r_1)$. Then, one of the following holds, where M is the optimal edit distance mapping:*

- *u does not appear in M,*
- *$(u, v) \in M$ for some child v of r_2, where $dist(T_1(u), T_2(v)) \geq 1$,*
- *$(u, v) \in M$ for some descendant v of some child of r_2, where $dist(T_1(u), T_2(v)) \geq 1$.*

(Proof) Since deletion of the root node is not allowed due to the definition, $(u, r_2) \in M$ never occurs. Therefore, the above three cases cover all the possible cases. For the second case, $dist(T_1(u), T_2(v)) \geq 1$ follows from the assumption. For the third case, $dist(T_1(u), T_2(v)) \geq 1$ holds because $|T_1(u)|$ is the largest and thus $|T_1(u)| > |T_2(v)|$ holds. □

Although the statement of the following lemma is simple, the proof is involved. Thus, readers can skip the proof at first and then return to the proof if interested in the details.

Lemma 2 *Let $x_1, \ldots, x_{l_1}$ and $y_1, \ldots, y_{l_2}$ be the children of $r(T_1)$ and $r(T_2)$, respectively. If $T_1(x_i) \approx T_2(y_j)$ holds, $dist(T_1, T_2) = dist(T_1 - T_1(x_i), T_2 - T_2(y_j))$ holds.*

In order to prove this lemma, we begin with a special case.

Lemma 3 *Suppose that x_i corresponds to a descendant y_0 of $r_2 = r(T_2)$ and y_j corresponds to a descendant x_0 of $r_1 = r(T_1)$ in the edit distance mapping M between T_1 and T_2. If $T_1(x_i) \approx T_2(y_j)$, $dist(T_1, T_2) = dist(T_1 - T_1(x_i), T_2 - T_2(y_j))$ holds.*

(Proof) Since $dist(T_1(x_i), T_2(y_j)) = 0$ holds from $T_1(x_i) \approx T_2(y_j)$, $dist(T_1, T_2) \leq dist(T_1 - T_1(x_i), T_2 - T_2(y_j))$ holds. Thus, we will prove that $dist(T_1, T_2) \geq dist(T_1 - T_1(x_i), T_2 - T_2(y_j))$ holds.

When x_i corresponds to y_j in M (i.e., $y_0 = y_j$), M includes an isomorphic mapping between $T_1(x_i)$ and $T_2(y_j)$ because the distance could otherwise be decreased by replacing a mapping between $T_1(x_i)$ and $T_2(y_j)$ with an isomorphic mapping. Therefore, the lemma holds from

$$dist(T_1, T_2) = dist(T_1 - T_1(x_i), T_2 - T_2(y_j)).$$

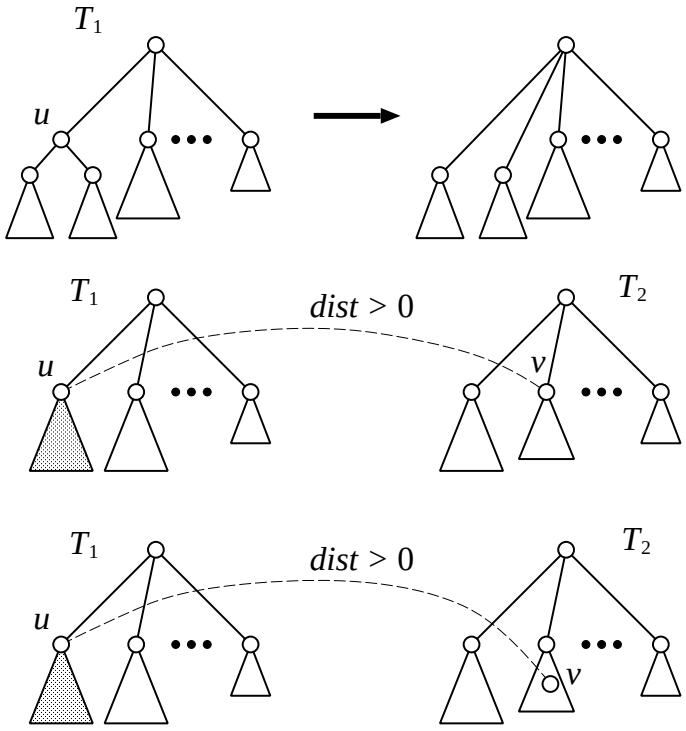

Figure 2: Three cases considered in Lemma 1.

Hereafter, we assume w.l.o.g. $x_0 \neq x_i$ and $y_0 \neq y_j$. Since x_i corresponds to y_0 and x_i is a child of r_1, y_0's ancestors but r_2 cannot appear in M. Similarly, x_0's ancestors but r_1 cannot appear in M. Thus, we can assume w.l.o.g. that $dist(T_1, T_2)$ can be represented as

$$dist(T_1, T_2) \;=\; dist(T_1(x_i), T_2(y_0)) \;+\; dist(T_1(x_0), T_2(y_j)) \;+\; d_0,$$

where d_0 is some appropriate value. Then, we can see

$$\begin{aligned} dist(T_1, T_2) &= dist(T_1(x_i), T_2(y_0)) + dist(T_1(x_0), T_2(y_j)) + d_0 \\ &= dist(T_2(y_j), T_2(y_0)) + dist(T_1(x_0), T_2(y_j)) + d_0 \\ &\geq dist(T_1(x_0), T_2(y_0)) + d_0 \\ &= dist(T_1(x_0), T_2(y_0)) + dist(T_1(x_i), T_2(y_j)) + d_0, \end{aligned}$$

where the second equality comes from $T_1(x_i) \approx T_2(y_j)$, the third inequality comes from the triangle inequality, and the fourth equality comes from
$dist(T_1(x_i), T_2(y_j)) = 0$. Since the change of mappings among $T_1(x_i)$, $T_1(x_0)$, $T_2(y_j)$ and $T_2(y_0)$ does not affect the other parts (see also Fig. 3), we can also obtain a mapping M' between T_1 and T_2 in which x_i corresponds to y_j. Therefore, $dist(T_1, T_2) \geq dist(T_1 - T_1(x_i), T_2 - T_2(y_j))$ holds. □

In the above case, the whole part of $T_1(x_i)$ corresponds to some subtree in T_2. However, we need to consider the case where a part of $T_1(x_i)$ corresponds to a part of $T_2(y_j)$ and the other parts of $T_1(x_i)$ correspond to other parts of T_2 (not in $T_2(y_j)$). By considering such a case, we can prove Lemma 2 as below.

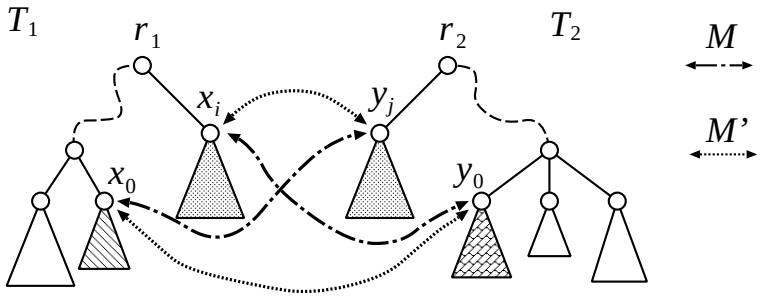

Figure 3: Explanation of Lemma 3.

Proof of Lemma 2

We prove the lemma by means of induction on the size of $T_1(x_i) \approx T_2(y_j)$.

First, we consider the case of size 1. If both x_i and y_j appear in the edit distance mapping, the lemma directly follows from Lemma 3. Otherwise, we can modify the mapping without increasing the distance by deleting mappings for x_i and y_j (if any) and adding a mapping between x_i and y_j.

Next, we assume that the lemma holds when the size of isomorphic subtrees is less than m. As in Lemma 3, $dist(T_1, T_2) \leq dist(T_1 - T_1(x_i), T_2 - T_2(y_j))$ holds. Thus, we will prove that $dist(T_1, T_2) \geq dist(T_1 - T_1(x_i), T_2 - T_2(y_j))$ holds. Now we assume w.l.o.g. that $T_1(x_1) \approx T_2(y_1)$ holds and the size of these subtrees are m. Then, subtrees rooted at some nodes in $T_1(x_1)$ correspond to subtrees rooted at some nodes in $T_2(y_1)$, where some nodes in the subtrees can be inserted, deleted or substituted. We only show the case where only one subtree in $T_1(x_1)$ corresponds to a subtree in $T_2(y_1)$. Extension to the cases with no such subtrees and with multiple subtrees is straight-forward. We also need to consider the case where y_1 (resp., x_1) corresponds to a descendant of x_1 (resp., y_1). However, such a case can be treated by letting $c = 1$ (resp., $b = 1$) in the following.

Let x'_1 and y'_1 be the roots of such subtrees (see also Fig. 4). Let subtrees rooted at $x'_2, \ldots, x'_b$ in $T_1(x_1)$ correspond to subtrees rooted at $y_2, \ldots, y_b$ in T_2 but not in $T_2(y_1)$, respectively. Let subtrees rooted at $y'_2, \ldots, y'_c$ in $T_2(y_1)$ correspond to subtrees rooted at $x_2, \ldots, x_c$ in T_1 but not in $T_1(x_1)$, respectively. We can assume w.l.o.g. that $x_2, \ldots, x_c, y_2, \ldots, y_b$ are children of the roots because their ancestors but the roots cannot correspond to any nodes.

Next, we make a copy of $T_1(x'_1)$ and let it be a child of r_1. Similarly, we make a copy of $T_1(x'_1)$ and let it be a child of r_2. Let x_{c+1} and y_{b+1} be the roots of these subtrees and T'_1 and T'_2 be the resulting subtrees, respectively. It should be noted that there can be other children $x_{c+2}, x_{c+3}, \ldots$ and $y_{b+2}, y_{b+3}, \ldots$ of r_1 and r_2, respectively. From the induction hypothesis, we have

$$dist(T_1, T_2) = dist(T'_1, T'_2).$$

Then, we add a new node x_0 as a child of r_1 and make $x_2, \ldots, x_{c+1}$ be the children of x_0. Similarly, we add a new node y_0 as a child of r_2 and make $y_2, \ldots, y_{b+1}$ be the children of y_0. We give identical labels α not appearing in the other nodes to x_0 and y_0. Let T''_1 and T''_2 be the resulting trees. By carrying over the edit distance mapping between T_1 and T_2 except for $T'_1(x'_1)$ and $T'_2(y'_1)$, transferring mapping between $T'_1(x'_1)$ and $T'_2(y'_1)$ to mapping between $T'_1(x_{c+1})$ and $T'_2(y'_1)$, adding isomorphic mapping between $T'_1(x'_1)$ and $T'_2(y_{b+1})$, and adding mapping between x_0 and y_1, and between x_1 and y_0, we have a mapping between T''_1 and T''_2.

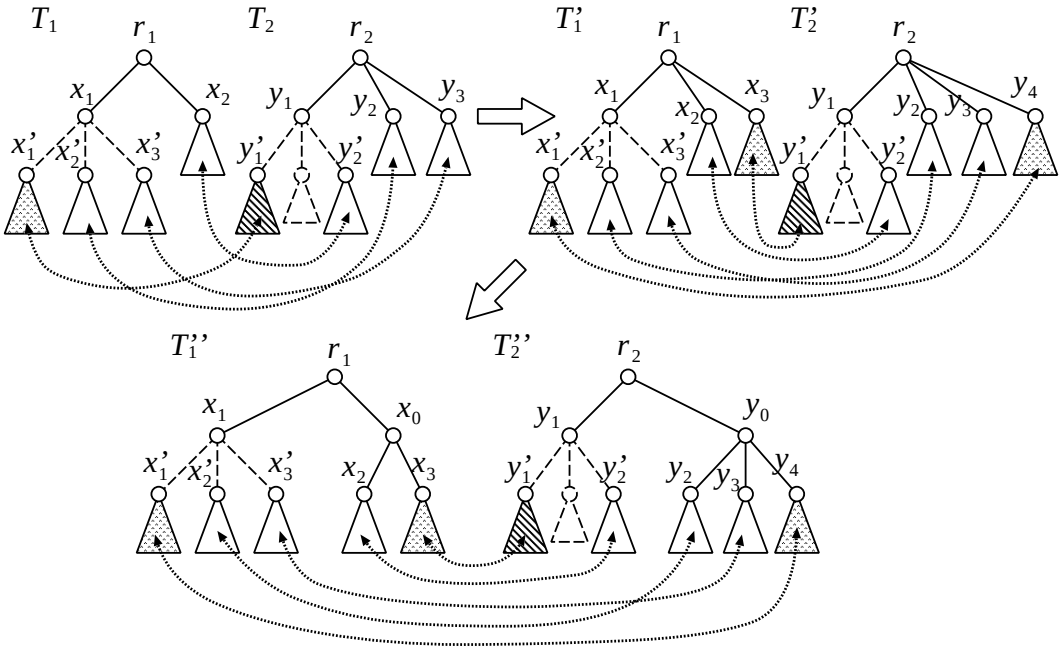

Figure 4: Explanation of Lemma 2. In this case, $b = 3$ and $c = 2$. A partial mapping between $T_1'(x_1')$ and $T_2'(y_1')$ is replaced by an isomorphic mapping between $T_1'(x_1')$ and $T_2'(y_4)$ and a partial mapping between $T_1'(x_3)$ and $T_2'(y_1')$.

From the mapping, we have

$$\begin{aligned} dist(T_1, T_2) &= dist(T_1(x_1'), T_2(y_1')) + \sum_{i=2}^{b} dist(T_1(x_i'), T_2(y_i)) \\ &\quad + \sum_{j=2}^{c} dist(T_1(x_j), T_2(y_j')) + d_0 + d_1 + d_2 \\ &= dist(T_1', T_2') \\ &\geq dist(T_1''(x_1), T_2''(y_0)) + dist(T_1''(x_0), T_2''(y_1)) + d_0, \end{aligned}$$

where d_1 denotes the cost for deleted nodes in $T_1(x_1) - T_1(x_1') - T_1(x_2') - \cdots - T_1(x_b')$, d_2 denotes the cost for inserted nodes in $T_2(y_1) - T_2(y_1') - T_2(y_2') - \cdots - T_2(y_c')$, d_0 corresponds to the other costs not counted in other terms in the right hand side of the first line. It is to be noted that x_0 and x_1 are mapped to y_1 and y_0 respectively in the last expression, which does not increase the distance because x_1 and y_1 (and their ancestors) do not appear in the edit distance mapping and

$$\gamma(\ell(x_0), \ell(y_1)) + \gamma(\ell(x_1), \ell(y_0)) = 2 = \gamma(\ell(x_1), \epsilon) + \gamma(\epsilon, \ell(y_1))$$

holds [4].

[4] Even for non-unit cost cases, the lemma holds by letting $\gamma(e, \alpha) = \gamma(e, \epsilon)$ for all $e \in \Sigma'$.

Since x_1 and y_1 are mapped to y_0 and x_0 respectively and $T_1''(x_1) \approx T_2''(y_1)$ holds, as in the proof of Lemma 3, we have

$$dist(T_1''(x_1), T_2''(y_0)) + dist(T_1''(x_0), T_2''(y_1)) + d_0 \quad \geq$$
$$dist(T_1''(x_0), T_2''(y_0)) + dist(T_1''(x_1), T_2''(y_1)) + d_0.$$

From the right hand side of the above inequality, we obtain a mapping M' between T_1'' and T_2''. Then, we can assume w.l.o.g. that x_0 corresponds to y_0 in M' because $\gamma(\ell(x_0), \ell(y_0)) = \gamma(\alpha, \alpha) = 0$. By the induction hypothesis, we can also assume that $T_1''(x_{c+1})$ corresponds to $T_2''(y_{b+1})$ in the edit distance mapping giving $dist(T_1''(x_0), T_2''(y_0))$.

We recover a mapping between T_1 and T_2 from M' by deleting a partial mapping between x_0 and y_0 and deleting a partial mapping between $T_1''(x_{c+1})$ and $T_2''(y_{b+1})$. This does not increase the distance because x_0 and y_0 have the identical label and $T_1''(x_{c+1}) \approx T_2''(y_{b+1})$. Therefore, we recover a mapping between T_1 and T_2 such that $T_1(x_1)$ corresponds to $T_2(y_1)$ and it gives the distance at most $dist(T_1, T_2)$. This completes the proof of the lemma. □

Next, we present an $O(2.62^k \cdot poly(n))$ time algorithm. Before presenting it, we begin with an $O(2^k \cdot k! \cdot poly(n))$ time algorithm because it is easier to understand.

This simpler algorithm decides in a bottom up manner whether or not $dist(T_1(u), T_2(v)) \leq h$ holds for all $(u, v, h) \in V(T_1) \times V(T_2) \times \{0, 1, 2, \cdots, k\}$ under a condition that u corresponds to v. Consider the case of $u = r(T_1)$ and $v = r(T_2)$, where the other cases can be processed in the same way. Then, the algorithm maintains two trees that are obtained by recursively deleting some children of the roots. It also maintains W_1 and W_2 that are candidates of matching pairs. Let $x_1, \ldots, x_{d_1}$ and $y_1, \ldots, y_{d_2}$ be the children of u and v, respectively. If $T_1(x_i) \approx T_2(y_j)$ holds for some (i, j), we can delete $T_1(x_i)$ and $T_2(y_j)$ without affecting the resulting distance because of Lemma 2. Otherwise, let w be the child such that $|T(w)|$ is the largest, where we assume w.l.o.g. that $T = T_1$. Then, from Lemma 1, w should be deleted or matched with some non-root node of T_2. For the former case, we examine whether or not $dist(T_1 - w, T_2) \leq k - 1$ holds because we have already used one edit operation to delete w. Otherwise, w must be matched with some node of T_2 and thus we put w into W_1. We repeat this procedure as long as there exist a child not in $W_1 \cup W_2$ and $|W_1| \leq k$ and $|W_2| \leq k$ hold. Finally, if $|W_1| > k$ or $|W_2| > k$ holds, we can conclude $dist(T_1, T_2) > k$. Otherwise, we computes an optimal one-to-one mapping between W_1 and W_2 if it exists (i.e., $|W_1| = |W_2|$).

The following is a pseudocode of this simpler algorithm. In this procedure, we let $mindist(u, v) = \min_h\{FpDist0(T_1(u), T_2(v), h) = TRUE\}$, where $mindist(u, v) = \infty$ if $FpDist0(T_1(u), T_2(v), h) = FALSE$ for all h. We can assume that this value can be computed before computation of $FpDist0(T_1, T_2, k)$ by execution of $FpDist0(T_1(u), T_2(v), h)$ for $h = 0, \ldots, k$ for all $(u, v) \in des(r(T_1)) \times des(r(T_2))$ in a bottom up manner (i.e., results of $FpDist0(T_1(u), T_2(v), h)$ are stored in a DP table). We can also assume $\ell(r(T_1)) = \ell(r(T_2))$. Otherwise, it is enough to execute $FpDist0(T_1, T_2, k - 1)$ ($FpDist0(T_1, T_2, k - \gamma(\ell(r(T_1)), \ell(r(T_2))))$ for the integer cost model) instead of $FpDist0(T_1, T_2, k)$.

Procedure $FpDist0(T_1, T_2, k)$
if $k < 0$ **then return** FALSE;
if $|T_1| = 0$ or $|T_2| = 0$ **then**
  if $\max(|T_1|, |T_2|) \leq k$ **then return** TRUE **else return** FALSE;
if $|T_1| = |T_2| = 1$ **then**
  if $\gamma(\ell(r(T_1)), \ell(r(T_2))) \leq k$ **then return** TRUE **else return** FALSE;
if $T_1(x_i) \approx T_2(y_j)$ for some $x_i \in chd(r(T_1))$, $y_j \in chd(r(T_2))$ **then**

return $FpDist0(T_1 - T_1(x_i), T_2 - T_2(y_j), k)$;
$W_1 \leftarrow \emptyset$; $W_2 \leftarrow \emptyset$;
while $|W_1| \leq k$ and $|W_2| \leq k$ **do**
  Let w be the node in $chd(r_1) \cup chd(r_2) - W_1 - W_2$ such that
    $|T(w)|$ is the largest; (T is either T_1 or T_2)
  if such a node does not exist **then** (#1)
    if $|W_1| \neq |W_2|$ **then return** FALSE;
    Compute the minimum weight bipartite matching between W_1 and W_2
      where $weight(u, v) = mindist(u, v)$;
    if the minimum cost $\leq k$ **then return** TRUE **else return** FALSE;
  if $w \in chd(r(T_1))$ and $FpDist0(T_1 - w, T_2, k-1)$ is TRUE
  then return TRUE; (#2)
  if $w \in chd(r(T_2))$ and $FpDist0(T_1, T_2 - w, k-1)$ is TRUE
  then return TRUE; (#3)
  if $w \in chd(r(T_1))$ **then** $W_1 \leftarrow W_1 \cup \{w\}$ **else** $W_2 \leftarrow W_2 \cup \{w\}$;
return FALSE;

Theorem 1 *$FpDist0(T_1, T_2, k)$ decides whether or not $dist(T_1, T_2) \leq k$ holds in $O(2^k \cdot k! \cdot poly(n))$ time and $O(n^2)$ space.*

(Proof) First, we show the correctness of the algorithm. If either $|T_1| = 0$, $|T_2| = 0$ or $|T_1| = |T_2| = 1$ holds, the algorithm obviously returns the correct value (TRUE or FALSE). If $T_1(x_i) \approx T_2(y_j)$ holds, the correctness follows from induction on the total size of trees and Lemma 2. Otherwise, $dist(T_1(x_i), T_2(y_j)) \geq 1$ must hold for all pairs (x_i, y_j).

Suppose that part (#1) is executed. Then, all the children of $r(T_1)$ and $r(T_2)$ are included in $W_1 \cup W_2$. Since deletion of any child should have been taken care by (#2) and (#3), FALSE should be returned if $|W_1| \neq |W_2|$ holds. Otherwise, there must exist a one-to-one mapping between $chd(r(T_1))$ and $chd(r(T_2))$. Then, $dist(T_1, T_2) \leq k$ holds if and only if the weight of the minimum weight matching is at most k.

Suppose that some of the children of $r(T_1)$ or $r(T_2)$ is deleted in the optimal mapping, then such a node should have been correctly taken care by (#2) or (#3).

Suppose that $|W_1| > k$ or $|W_2| > k$ holds. We can assume w.l.o.g. that $|W_1| > k$ holds. Since we can assume that no node in $W_1 \cup W_2$ is deleted, each node u in W_1 is mapped to some node in W_2, or some child v of $r(T_2)$ or its descendant such that $v \notin W_2$. If u is mapped to a node in W_2, it contributes to the total distance by at least 1 because all isomorphic pairs $(T_1(x_i), T_2(y_j))$s are removed beforehand. Otherwise, $|T_1(u)| \geq |T_2(v)|$ holds and thus we can see from Lemma 1 that u contributes to the total distance by at least 1. Thus, FALSE should be output in this case. This completes the proof of the correctness of $FpDist0(T_1, T_2, k)$.

Next, we analyze the time complexity. Let $f(k, n)$ be the time complexity of $FpDist0(T_1, T_2, k)$, where $n = \max(|T_1|, |T_2|)$. If either $|T_1| = 0$, $|T_2| = 0$ or $|T_1| = |T_2| = 1$ holds, $FpDist0(T_1, T_2, k)$ takes $O(1)$ time. $T_1(x_i) \approx T_2(y_j)$ can be tested in $O(1)$ time per pair if we pre-process T_1 and T_2 in linear time so that the signature (with $O(\log n)$ bits) of each $T_i(v)$ is computed [4]. Therefore, whether or not such a pair exists can be tested in $O(n^2)$ time. Execution of the while loop can be done in $O(kn)$ time in total except minimum weight matching and recursive call of $FpDist0$. Minimum weight bipartite matching can be done in $O(k^3)$ ($\leq O(n^3)$) time [2]. Therefore, we have $f(k, n) \leq 2k \cdot f(k-1, n) + O(n^3)$. From this, we can show that $f(k, n)$

is $O(2^k \cdot k! \cdot n^3)$ as follows:

$$\begin{aligned} f(k,n) &\leq O(n^3) \cdot \left[1 + 2k + 2k \cdot 2(k-1) + 2k \cdot 2(k-1) \cdot 2(k-2) + \cdots + 2^k \cdot k!\right] \\ &\leq O(n^3) \cdot \left[(1 + 2 + 2^2 + \cdots + 2^k) \cdot k!\right] \leq O(n^3) \cdot \left[2^{k+1} \cdot k!\right], \end{aligned}$$

where we can assume that $f(0,n)$ is $O(n)$ because the tree isomorphism problem can be solved in $O(n)$ time [4]. Here, it should be noted that $FpDist0$ should be executed $k+1$ ($\leq O(n)$) times for $O(n^2)$ pairs. Therefore, the total time complexity is $O(2^k \cdot k! \cdot n^6)$.

Finally, we analyze the space complexity. For each pair of $T_1(u)$ and $T_2(v)$, we need to keep $dist(T_1(u), T_2(v))$ up to k, from which we can know the results of $FpDist0(T_1(u), T_2(v), h)$ for $h = 0, \ldots, k$. Therefore it uses $O(n^2)$ space. For each depth of recursive call of $FpDist0(T_1, T_2, k)$, we need $O(n)$ space. Therefore, in total, $O(kn)$ ($\leq O(n^2)$) space is required. Since $O(n^2)$ space is enough for the minimum weight bipartite matching, the total space complexity is $O(n^2)$. □

Although the polynomial factor in Theorem 1 might be considerably reduced by more careful analysis, we do not perform such an analysis because reduction of exponential factors on k is much more important than that of polynomial factors.

Now, we present our main algorithm which we call $FpDist$. This algorithm is almost the same as the original $FpDist0$ except that we inherit the sets W_1 and W_2 of the caller. By inheriting the sets W_1 and W_2, we can reduce the number of iterations in the while loops. The following is the pseudo code of $FpDist0$, where it is invoked as $FpDist(T_1, T_2, k, \emptyset, \emptyset)$.

Procedure $FpDist(T_1, T_2, k, W_1, W_2)$
if $k < 0$ **then return** FALSE;
if $|T_1| = 0$ or $|T_2| = 0$ **then**
  if $\max(|T_1|, |T_2|) \leq k$ **then return** TRUE **else return** FALSE;
if $|T_1| = |T_2| = 1$ **then**
  if $\gamma(\ell(r(T_1)), \ell(r(T_2))) \leq k$ **then return** TRUE **else return** FALSE;
if $T_1(x_i) \approx T_2(y_j)$ for some $x_i \in chd(r(T_1))$, $y_j \in chd(r(T_2))$ **then**
  return $FpDist(T_1 - T_1(x_i), T_2 - T_2(y_j), k, W_1, W_2)$; (*)
while $|W_1| \leq k$ and $|W_2| \leq k$ **do**
  Let w be the node in $chd(r_1) \cup chd(r_2) - W_1 - W_2$ such that
    $|T(w)|$ is the largest; (T is either T_1 or T_2)
  if such a node does not exist **then** (#1)
    if $|W_1| \neq |W_2|$ **then return** FALSE;
    Compute the minimum weight bipartite matching between W_1 and W_2
      where $weight(u,v) = mindist(u,v)$;
    if the minimum cost $\leq k$ **then return** TRUE **else return** FALSE;
  if $w \in chd(r(T_1))$ and $FpDist(T_1 - w, T_2, k-1, W_1, W_2)$ is TRUE
  then return TRUE; (#2)
  if $w \in chd(r(T_2))$ and $FpDist(T_1, T_2 - w, k-1, W_1, W_2)$ is TRUE
  then return TRUE; (#3)
  if $w \in chd(r(T_1))$ **then** $W_1 \leftarrow W_1 \cup \{w\}$ **else** $W_2 \leftarrow W_2 \cup \{w\}$;
return FALSE;

The correctness of $FpDist$ follows from the fact that each call to $FpDist(T_1, T_2, k, W_1, W_2)$ satisfies the following conditions.

(a) The size of subtree $T(w)$ for $w \in W_1 \cup W_2$ is greater than or equal to that of $T(w')$, $w' \in chd(r_1) \cup chd(r_2) - W_1 - W_2$.

(b) If x_i and y_j satisfy the condition of the if-statement at (*), then we have $x_i \notin W_1$ and $y_j \notin W_2$.

We assume that all ties in choosing w in the while loop are broken consistently, for example, by using an arbitrary total order on $V(T_1) \cup V(T_2)$. The conditions are obvious if both W_1 and W_2 are empty. Otherwise, we can use mathematical induction to prove them. If the condition (a) is satisfied in the beginning of *FpDist*, we can see that each recursive call satisfies the condition (a) as follows. Among the three recursive calls, the first one satisfies (a) since the remaining subtrees and the sets W_1 and W_2 do not change, the latter two calls satisfy (a) since the subtrees appended by deleting w are strictly smaller than $T(w)$, and the members of W_1 and W_2 are appended according to the size of subtrees. Therefore, the condition (a) is satisfied everywhere in *FpDist*. Next, we prove that *FpDist* satisfies the condition (b). The case when $W_1 = W_2 = \emptyset$ is obvious. If *FpDist* is called at (*) of the caller, the condition (b) follows directly from the condition (b) of the caller. We assume without loss of generality that *FpDist* is called at (#2). Let w be the node which is deleted by the caller. Suppose that $T_1(x_i) \approx T_2(y_j)$ holds for some x_i and y_j. If $x_i \notin chd(w)$, the pair $T_1(x_i)$ and $T_2(y_j)$ must have been deleted by the caller, which contradicts to the assumption. If $x_i \in chd(w)$, we have $|T_2(y_j)| = |T_1(x_i)| < |T_1(w)|$. Therefore we have $x_i \notin W_1$ and $y_j \notin W_2$ since W_1 and W_2 are constructed according to the size of subtree. Thus *FpDist* satisfies the condition (b).

Next, we analyze the time complexity of *FpDist*. Let $g(k, n, s)$ denote the time complexity of $FpDist(T_1, T_2, k, W_1, W_2)$, where $n = \max(|T_1|, |T_2|)$ and $s = |W_1| + |W_2|$. The total time complexity is given by $g(k, n, 0)$. There exists a polynomial $G(n)$ that satisfies $g(k, n, s) \leq G(n)$ for $s > 2k$ since the algorithm does not execute the while loop in such a case. For the other cases, we have the following inequality:

$$g(k, n, s) \leq g(k-1, n, s) + g(k-1, n, s+1) + \cdots + g(k-1, n, 2k) + T(n),$$

where $T(n)$ is another polynomial which satisfies $T(n) = O(n^3)$. We can prove $g(k, n, 2k-i) \leq F_{i+2} \cdot G(n) + F_{i+1} \cdot T(n)$, where F_i is the i-th Fibonacci number, by mathematical induction.

$$\begin{aligned}
g(k, n, 2k) &\leq g(k-1, n, 2k) + T(n) \leq G(n) + T(n), \\
g(k, n, 2k-1) &\leq g(k-1, n, 2k-1) + g(k-1, n, 2k) + T(n) \leq 2G(n) + T(n), \\
g(k, n, 2k-i) &\leq \sum_{j=0}^{i} g(k-1, n, 2k-j) + T(n) \\
&\leq \sum_{j=2}^{i} g(k-1, n, 2k-j) + 2G(n) + T(n) \\
&= \sum_{j=0}^{i-2} g(k-1, n, 2(k-1)-j) + 2G(n) + T(n) \\
&\leq \sum_{j=0}^{i-2} \{F_{j+2} \cdot G(n) + F_{j+1} \cdot T(n)\} + 2G(n) + T(n) \\
&= F_{i+2} \cdot G(n) + F_{i+1} \cdot T(n).
\end{aligned}$$

The last equality comes from $\sum_{j=0}^{i} F_j = F_{i+2} - 1$. Hence the total time complexity is $g(k, n, 0) = F_{2k+2} \cdot G(n) + F_{2k+1} \cdot T(n) = O(2.62^k \cdot poly(n))$ since $F_{2k} \leq \frac{1}{\sqrt{5}} \cdot (\frac{1+\sqrt{5}}{2})^{2k} = O(2.62^k)$.

Analysis of the space complexity can be done in the same way as in the proof of Theorem 1.

Theorem 2 *$FpDist(T_1, T_2, k)$ decides whether or not $dist(T_1, T_2) \leq k$ holds in $O(2.62^k \cdot poly(n))$ time and $O(n^2)$ space.*

It is to be noted that we did not allow deletion of a root because it converts a tree into a forest [1]. However, we can easily cope with the case where the root in one tree corresponds to a non-root node in the other tree.[5] In such a case, it is enough to compute $dist(T_1, T_2(v)) + \sum_{v' \in V(T_2) - V(T_2(v))} \gamma(\epsilon, \ell(v'))$ and $dist(T_1(u), T_2) + \sum_{u' \in V(T_1) - V(T_1(u))} \gamma(\ell(u'), \epsilon))$ for all $u \in V(T_1) - \{r(T_1)\}$ and $v \in V(T_2) - \{r(T_2)\}$. However, $dist(T_1, T_2(v))$s and $dist(T_1(u), T_2))$s are also obtained by the fixed-parameter algorithms (up to distance k). Therefore, Theorems 1 and 2 still hold even if one root corresponds to a non-root node,

Finally, we show that the algorithms presented in this section can be extended for the case in which costs of edit operations are integers and $\gamma(a, b) > 0$ holds for all $a \neq b$. For this purpose, it is enough to replace $k-1$ in (#2) and (#3) (in both algorithms) with $k - \gamma(\ell(w), \epsilon)$. Since the distance under this integer cost model is always greater than or equal to that under the unit cost model, all the proofs in this section are valid (as they are) even if we replace $\gamma(a, b) = 1$ for all $a \neq b$ with $\gamma(a, b) \geq 1$ for all $a \neq b$. Therefore, Theorems 1 and 2 hold for the integer cost model.

4 Algorithms for Bounded Degree LCST

Though the edit distance problem is NP-hard for unordered trees, we can obtain an exact solution in polynomial time if the maximum degree of the corresponding LCST (i.e., LCST obtained from an optimal mapping) is bounded by a constant. In this section, we first present a basic algorithm and then present improved algorithms, where we only need to assume that the distance satisfies the conditions on a distance metric.

4.1 Basic Algorithm

The basic algorithm is quite simple and is based on a simple DP procedure. As explained in Section 2, computation of the edit distance is equivalent to computation of the LCST. Therefore, we focus on computation of the LCST in this section. For $x \in V(T_1)$ and $y \in V(T_2)$, let $S_D(x, y)$ be the size of LCST between $T_1(x)$ and $T_2(y)$ under the condition that the maximum outdegree (i.e., maximum $deg(v)$) of LCST is at most D. Though the roots need not correspond to each other in the following, we can modify the algorithms so that the roots correspond to each other by letting $f(r(T_1), r(T_2))$ be very large.

For the simplicity, we begin with the case of $D = 2$. In this case, each node has at most two children in an LCST. Therefore, to compute $S_2(x, y)$, it is enough to examine combinations of at most two descendants of x and y, respectively. The following is a DP procedure of the

[5] It does not happen that both roots are deletes because the distance could have reduced if deletions of two roots were replaced by a substitution.

algorithm.

$$S_2(x,y) = \max \begin{cases} f(x,y), & -(*1) \\ \max_{x_1,x_2 \in des(x), y_1,y_2 \in des(y)} \{S_2(x_1,y_1) + S_2(x_2,y_2) \\ \qquad + f(x,y)\}, & -(*2) \\ \max_{x_1 \in des(x), y_1 \in des(x)} \{S_2(x_1,y_1) + f(x,y)\}, & -(*3) \\ \max_{y_1 \in des(y)} S_2(x,y_1), & -(*4) \\ \max_{x_1 \in des(x)} S_2(x_1,y), & -(*5) \end{cases}$$

where $x_1 \notin des(x_2) \cup \{x_2\}$, $x_2 \notin des(x_1) \cup \{x_1\}$, $y_1 \notin des(y_2) \cup \{y_2\}$ and $y_2 \notin des(y_1) \cup \{y_1\}$ must hold. It is to be noted that the maximum degree of T_1 and T_2 need not be bounded. Let $BdDist_2$ denote the above DP algorithm. Then, we have the following theorem.

Theorem 3 *$BdDist_2$ computes the edit distance in $O(n^6)$ time and $O(n^2)$ space if the maximum outdegree of the corresponding largest common subtree is at most 2.*

(Proof) We consider an optimal mapping M between $T_1(x)$ and $T_2(y)$. Then, either one of the following must hold:

(i) x corresponds to y,

(ii) x corresponds to a descendant of y,

(iii) y corresponds to a descendant of x.

It is to be noted that either x or y must appear in M. Otherwise, we can increase or keep the score of LCST by adding (x,y) to M since $f(x,y) \geq 0$ holds for any pair of nodes (x,y).

If x and y are leaves, LCST is clearly computed by (*1) and the other parts are not executed. Otherwise, cases (ii) and (iii) are covered by (*4) and (*5), respectively. For case (i), there are two possibilities:

- the root of $LCST(T_1(x), T_2(y))$ has two children,
- the root of $LCST(T_1(x), T_2(y))$ has only one child.

Then, the former case is covered by (*2) and the latter case is covered by (*3). Therefore, $BdDist_2$ correctly computes LCST if $D = 2$.

Next, we analyze the time complexity. Clearly, $S_2(x,y)$ must be computed for $O(n^2)$ pairs. For each pair, $O(n^4)$ combinations of four children are examined in (*2), where $O(1)$ time is enough per combination. Since (*2) is the most time consuming part, the total time complexity is $O(n^2) \times O(n^4) = O(n^6)$.

Finally, we analyze the space complexity. In order to store the values of $S_2(x,y)$ for all $x, y \in V(T_1) \times V(T_2)$. we need $O(n^2)$ space. Then, computation of each $S_2(x,y)$ can be done using constant space. Therefore, the total space complexity is $O(n^2)$. □

We can easily extend $BdDist_2$ for arbitrary fixed D. The following is the DP algorithm, which is denoted by $BdDist_D$.

$$S_D(x,y) = \max \begin{cases} \max_{h=0,\ldots,D} \left\{ \max_{x_1,\ldots,x_h \in des(x), y_1,\ldots,y_h \in des(y)} \left[\left(\sum_{i=1}^{h} S_D(x_i,y_i) \right) + f(x,y) \right] \right\}, \\ \max_{y_1 \in des(y)} S_D(x,y_1), \\ \max_{x_1 \in des(x)} S_D(x_1,y), \end{cases}$$

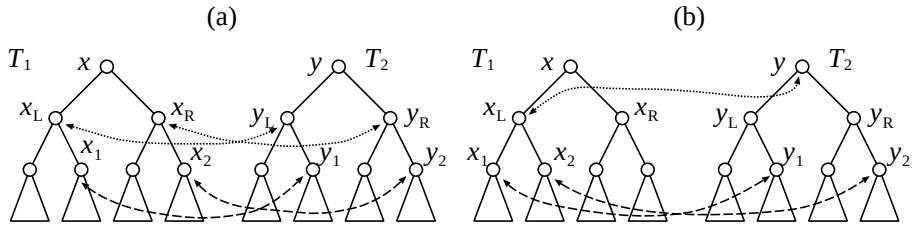

Figure 5: Illustration for explaining the key idea of the improved algorithm for $D = 2$.

where $x_i \notin des(x_j) \cup \{x_j\}$ and $y_i \notin des(y_j) \cup \{y_j\}$ must be satisfied for any $i \neq j$. Clearly, we have:

Corollary 1 *$BdDist_D$ computes the edit distance between two unordered trees in $O(n^{2+2D})$ time and $O(n^2)$ space if the maximum outdegree of the corresponding largest common subtree is at most D, where D is a constant.*

4.2 Improved Algorithm for $D = 2$

Though $BdDist_D$ works in polynomial time, it is not practical because the time complexity is $O(n^6)$ even for $D = 2$. In this subsection, we present an improved algorithm $FastBdDist_2$ for the case of $D = 2$ that works in $O(n^2)$ time.

In (*2) of $BdDist_2$, we examine all combinations of x_1, x_2, y_1, y_2. However, it is not necessary to examine all combinations. Suppose that x has two children x_L and x_R, and y has two children y_L and y_R (see Fig. 5(a)). Suppose also that x_1, x_2, y_1, y_2 are descendants of x_L, x_R, y_L, y_R, respectively. Then, we can see that $S_2(x_L, y_L) \geq S_2(x_1, y_1)$ and $S_2(x_R, y_R) \geq S_2(x_2, y_2)$ hold since $f(x, y) \geq 0$ holds for any node pair (x, y). Therefore, in such a case, $S_2(x, y)$ is given by $S_2(x_L, y_L) + S_2(x_R, y_R) + f(x, y)$.

We need to consider another case (see Fig. 5(b)). Suppose that x_1, x_2 are descendants of x_L, y_1 is a descendant of y_L, and y_2 is a descendant of y_R. If $(x, y) \notin M$ where M is an optimal mapping between $T_1(x)$ and $T_2(y)$, $S_2(x, y)$ is given by $S_2(x_L, y)$. Otherwise, (x, y) should have replaced (x_L, y) or (x'_L, y) in an optimal mapping between $T_1(x_L)$ and $T_2(y)$, where x'_L is a descendant of x_L and an ancestor of x_1 and x_2. Suppose that (x, y) replaced (x'_L, y). Then, $S_2(x, y)$ is given by $S_2(x'_L, y) - f(x'_L, y) + f(x, y)$. In order to cover this kind of cases, we keep the score of an uppermost pair giving $S_2(x, y)$.

Let $S_2^{++}(x, y)$ be the score of LCST with $D = 2$ under the condition that x corresponds to y. Let $S_2^{+-}(x, y)$ be the score of LCST with $D = 2$ under the condition that x corresponds to a descendant of y. Furthermore, $f^{+-}(x, y)$ denotes the score for x and the corresponding node. $S_2^{-+}(x, y)$ and $f^{-+}(x, y)$ are defined in an analogous way under the condition that y corresponds to a descendant of x. Then, we can compute $S_2(x, y)$ by the following DP procedure:

$$
\begin{aligned}
S_2(x, y) &= \max\{S_2^{++}(x, y), S_2^{+-}(x, y), S_2^{-+}(x, y)\}, \\
S_2^{++}(x, y) &= \max \begin{cases}
\max_{x_i, x_j \in chd(x), y_{i'}, y_{j'} \in chd(y)} S_2(x_i, y_{i'}) + S_2(x_j, y_{j'}) + f(x, y), \\
\max_{x_i \in chd(x), y_{i'} \in chd(y)} S_2(x_i, y_{i'}) + f(x, y), \\
\max_{y_i \in chd(y)} S_2^{++}(x, y_i) - f(x, y_i) + f(x, y), \\
\max_{x_i \in chd(x)} S_2^{++}(x_i, y) - f(x_i, y) + f(x, y),
\end{cases}
\end{aligned}
$$

$$S_2^{+-}(x,y) = \max \begin{cases} \max_{y_i \in chd(y)} S_2^{+-}(x,y_i), \\ \max_{y_i \in chd(y)} S_2^{++}(x,y_i), \end{cases}$$
$$S_2^{-+}(x,y) = \max \begin{cases} \max_{x_i \in chd(x)} S_2^{-+}(x_1,y), \\ \max_{x_i \in chd(x)} S_2^{++}(x_i,y), \end{cases}$$

where $x_i \neq x_j$ and $y_{i'} \neq y_{j'}$ must be satisfied. The initialization of the score table is straightforward. Computation of $f^{+-}(x,y)$ and $f^{-+}(x,y)$ can be done in a similar way as above.

It is straight-forward to see that the above algorithm correctly works in $O(n^4)$ time. We can further reduce the time complexity to $O(n^2)$ as follows. For each node pair (x,y), we construct a weighted bipartite graph with disjoint vertex sets $chd(x)$ and $chd(y)$ such that the weight of an edge between $x_i \in chd(x)$ and $y_j \in chd(y)$ is given by $S_2(x_i,y_j)$. Then, computation of

$$\max_{x_i,x_j \in chd(x), y_{i'},y_{j'} \in chd(y)} S_2(x_i,y_{i'}) + S_2(x_j,y_{j'})$$

can be reduced to computation of the maximum weight matching using two edges. This can be done in $O(deg(x)deg(y))$ time as follows. Let (x_p,y_q) be the pair with the maximum weight. Let $(x_{p'},y_{q'})$ be the pair with the maximum weight after removing x_p and y_q. We compute $S_p = \max_{y_j \in chd(y), y_j \neq y_q} S_2(x_p,y_j)$ and $S_q = \max_{x_i \in chd(x), x_i \neq x_p} S_2(x_i,y_q)$. Then, we can obtain the desired maximum weight matching by computing $\max\{\ S_2(x_p,y_q)+S_2(x_{p'},y_{q'}),\quad S_p+S_q\ \}$. Therefore, $O(deg(x)deg(y))$ time is required per (x,y) and thus the time complexity of this improved algorithm ($FastBdDist_2$) is given by

$$O(\sum_{x \in V(T_1)} \sum_{y \in V(T_2)} deg(x)deg(y)) = O((n-1) \sum_{x \in V(T_1)} deg(x)) = O(n^2).$$

In order to store the score table, we need $O(n^2)$ space, which is enough for the other parts.

Theorem 4 *$FastBdDist_2$ computes the edit distance between two unordered trees in $O(n^2)$ time and $O(n^2)$ space if the maximum outdegree of the corresponding largest common subtree is at most 2.*

4.3 Algorithm Using Lowest Common Ancestor

It is very unclear whether or not $FastBdDist_2$ can be extended for the cases of $D > 2$. At least, complicated case analysis would be required and thus the extension is left as an open problem. Therefore, it is interesting to try to develop a simple and improved algorithm based on another idea.

In this subsection, we present an algorithm (called $LcaBdDist2_D$) using the *lowest common ancestor*, which works in $O(n^{2D})$ time for a fixed D. In order to determine the value of $S_D(x,y)$, $BdDist_D$ examines all possible consistent combinations of descendants of x and y under the degree constraint D, where we say that a tuple $(x_1,\ldots,x_h)$ is *consistent* if $x_i \notin des(x_j) \cup \{x_j\}$ holds for all $i \neq j$. On the contrary, each combination can be used for determining the value of $S_D(x',y')$s for all pairs $(x',y') \in (anc(x) \cup \{x\}) \times (anc(y) \cup \{y\})$. Formally, we have the following proposition, which directly follows from the definition of tree edit distance mapping.

Proposition 1 *If $(x_1,y_1),(x_2,y_2),\ldots,(x_h,y_h)$ are the children of (x,y) in the LCST, x and y are common ancestors of $x_1,x_2,\ldots,x_h$ and $y_1,y_2,\ldots,y_h$, respectively.*

Based on this proposition and the above discussions, we can compute $S_D(x, y)$s by examining all possible pairs of consistent tuples of $(x_1, x_2, \ldots, x_h)$ and $(y_1, x_2, \ldots, y_h)$ with updating the scores of their common ancestors. The following is a pseudocode of this algorithm, where $LCA(x)$ is defined as $p(x)$ (null if x is the root).

Procedure $LcaBdDist_D(T_1, T_2)$
for all $(x, y) \in V(T_1) \times V(T_2)$ **do** $S_D(x, y) \leftarrow f(x, y)$;
for all $h \in \{1, \ldots, D\}$ **do**
  for all consistent tuples $(x_1, \ldots, x_h)$ **do**
    $x_a \leftarrow LCA(x_1, \ldots, x_h)$;
    for all consistent tuples $(y_1, \ldots, y_h)$ **do**
      $y_a \leftarrow LCA(y_1, \ldots, y_h)$;
      for all (x, y) such that $x \in anc(x_a) \cup \{x_a\}$ and $y \in anc(y_a) \cup \{y_a\}$ **do**
        $S_D(x, y) \leftarrow \max\{S_D(x, y), S_D(x_1, y_1) + \cdots + S_D(x_h, y_h) + f(x, y)\}$;

In the above, the ordering of examining combinations is not specified. However, it must be carefully ordered so that $S_D(x_i, y_i)$ is used only after its final value is determined.

For that purpose, we sort combinations of $(x_1, \ldots, x_h)$ (resp., $(y_1, \ldots, y_h)$) as follows, where we identify each node with the number given by post order traversal of each tree, we denote $(x_1, \ldots, x_h) \prec (x'_1, \ldots, x'_{h'})$ if $(x_1, \ldots, x_h)$ is placed before $(x'_1, \ldots, x'_{h'})$ in L_0, and $L_1 \cdot L_2$ denotes the concatenation of lists L_1 and L_2,

Let L_0 be an empty list;
for $h = 1$ **to** D **do**
  Let L be a list of consistent tuples $(x_1, \ldots, x_h)$;
  Sort L alphabetically, where we identify each node as the number
    given by post order traversal;
  $L_0 \leftarrow L_0 \cdot L$;
Rearrange L_0 so that $\max\{x_1, \ldots, x_h\} \leq \max\{x'_1, \ldots, x'_{h'}\}$ holds
  for any tuple pair $(x_1, \ldots, x_h) \prec (x'_1, \ldots, x'_{h'})$;

Fig. 6 gives an example of ordering of tuples by this procedure.

Since $\max\{x_1, \ldots, x_h\} < LCA(x_1, \ldots, x_h)$ holds from the definition of LCA, we can see that $S_D(x, y)$ is used only after its final value is determined. Since alphabetical sorting can be done in linear time (for constant D) and rearrangement can also be done in linear time by simply selecting elements of L_0 from increasing order of $\max\{x_1, \ldots, x_h\}$, we can see that this procedure sorts all consistent tuples in time linear to the number of tuples and thus in $O(n^D)$ time (for each of T_1 and T_2).

Here, we analyze $LcaBdDist_D(T_1, T_2)$. The correctness of $LcaBdDist_D(T_1, T_2)$ directly follows from Proposition 1. Since LCA between two nodes can be computed in $O(1)$ time and $x_i \notin des(x_j) \cup \{x_j\}$ can be tested in $O(1)$ time [9], $O(1)$ time overhead is required per combination, where we assume that D is a constant. Therefore, the time complexity is $O(n^{2D+2})$ since there are $O(n^{2D})$ combinations and $O(n^2)$ ancestor pairs are examined per combination. Since we maintain the sorted lists of $O(n^D)$ tuples and $O(n^2)$ space is enough for the other purposes, the space complexity is $O(n^D + n^2)$.

We can reduce an $O(n^2)$ factor of the time complexity for updating the scores of ancestor pairs to $O(1)$. Observe that $LcaBdDist_D$ updates the scores of ancestor pairs of (x_a, y_a)

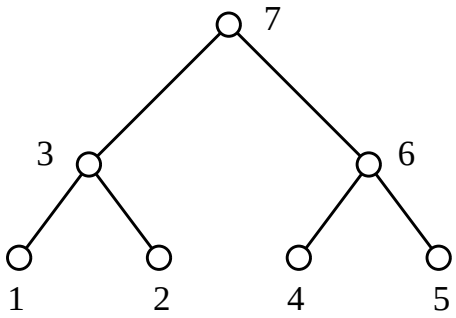

Figure 6: Example of ordering of tuples for $D = 2$. In this case, tuples are ordered as: (1) (2) (1,2) (3) (4) (1,4) (2,4) (3,4) (5) (1,5) (2,5) (3,5) (4,5) (6) (1,6) (2,6) (3,6) (7).

many times (i.e., for all tuple pairs $(x_1, \ldots, x_h)$ and $(y_1, \ldots, y_h)$ having the same LCA pair). However, it is enough to update the scores of ancestor pairs of (x_a, y_a) only when $S_D(x_a, y_a)$ is determined. For that purpose, we use an additional DP table $S_D^{--}(x, y)$ which stores the score of LCST between $T_1(x)$ and $T_2(y)$ under the condition that x corresponds to y but $f(x, y)$ is not counted in $S_D^{--}(x, y)$. This means that x or y can be mapped to another node in later updates. The following is a pseudocode for the improved algorithm using $S_D^{--}(x, y)$, where we use the same ordering as in $LcaBdDist$.

Procedure $LcaBdDist2_D(T_1, T_2)$
for all $(x, y) \in V(T_1) \times V(T_2)$ **do** $S_D(x, y) \leftarrow f(x, y)$; $S_D^{--}(x, y) \leftarrow 0$;
for all $h \in \{1, \ldots, D\}$ **do**
  for all consistent tuples $(x_1, \ldots, x_h)$ **do**
    $x_a \leftarrow LCA(x_1, \ldots, x_h)$;
    for all consistent tuples $(y_1, \ldots, y_h)$ **do**
      $y_a \leftarrow LCA(y_1, \ldots, y_h)$;
      if $h = 1$ **then**
        $S_D(x_1, y_1) \leftarrow \max\{S_D(x_1, y_1), S_D^{--}(x_1, y_1) + f(x_1, y_1)\}$;
        for all $x \in anc(x_1)$ and $y \in anc(y_1)$ **do**
          $S_D(x, y) \leftarrow \max\{S_D(x, y), S_D(x_1, y_1) + f(x, y)\}$;
        for all $x \in anc(x_1)$ **do**
          $S_D(x, y_1) \leftarrow \max\{S_D(x, y_1), S_D^{--}(x_1, y_1) + f(x, y_1)\}$;
        for all $y \in anc(y_1)$ **do**
          $S_D(x_1, y) \leftarrow \max\{S_D(x_1, y), S_D^{--}(x_1, y_1) + f(x_1, y)\}$;
      else
        $S_D^{--}(x_a, y_a) \leftarrow \max\{S_D^{--}(x_a, y_a), S_D(x_1, y_1) + \cdots + S_D(x_h, y_h)\}$;

The correctness of the improved algorithm directly follows from the definitions of $S_D(x, y)$ and $S_D^{--}(x, y)$ and the ordering (i.e., the scores of ancestor pairs of $S_D(x_1, y_1)$ are updated only after the value of $S_D(x_1, y_1)$ is determined. See also Fig. 7).

Since $O(n^{2D})$ tuple pairs are examined, update of $S_D^{--}(x, y)$ is performed $O(n^{2D})$ times. However, update of the scores of ancestor pairs is performed $O(n^2)$ times, each of which requires $O(n^2)$ time. Therefore, the total time complexity is $O(n^{2D} + n^4)$, which is $O(n^{2D})$ for $D \geq 2$. As in $LcaBdDist_D$. the space complexity is $O(n^D + n^2)$.

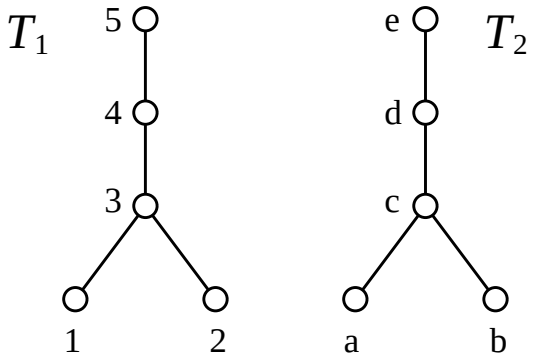

Figure 7: Example for explaining the difference between $LcaBdDist_D$ and $LcaBdDist2_D$. If a pair of ((1,2),(a,b)) or ((1,2),(b,a)) is examined, the scores of pairs in $S = \{$(3,c), (3,d), (3,e), (4,c), (4,d), (4,e), (5,c), (5,d), (5,e)$\}$ are updated in $LcaBdDist_D$, whereas only the score of (3,c) is updated in $LcaBdDist2_D$. In $LcaBdDist2_D$, the scores of pairs in S are updated when (3,c) is examined.

Theorem 5 *$LcaBdDist2_D$ computes the edit distance between two unordered trees in $O(n^{2D})$ time and $O(n^D)$ space if the maximum outdegree of the corresponding largest common subtree is at most D, where D is a constant such that $D \geq 2$.*

5 Concluding Remarks

We have presented an $O(2.62^k \cdot poly(n))$ time algorithm and an $O(n^{2D})$ time algorithm for the edit distance problem for unordered trees, where k is the maximum bound of the edit distance and D is the maximum degree of the largest common subtree. For the former algorithm, improvement of exponential factor is left as an open problem. However, the factor of 2.62^k is not large for moderate values of k. Therefore, it might be possible to develop a practical algorithm for comparing similar unordered trees based on this algorithm along with existing heuristics [10]. Such a development is left as future work. For the latter algorithm, it is unclear whether we can develop a fixed-parameter algorithm when D is regarded as a parameter. Therefore, deciding the complexity on D is left as an open problem.

Acknowledgements

We would like to thank an anonymous referee for CPM 2009 conference who pointed out a crucial error in a preliminary version of the fixed-parameter algorithm. The work of TA was partially supported by MEXT Grant-in-Aid No. 19650053 and No. 19200022. The work of DF and AT was partly supported by MEXT Grant-in-Aid No. 18049069.

References

[1] P. Bille, A survey on tree edit distance and related problem, Theoretical Computer Science 337 (2005) 217–239.

[2] T. H. Cormen, C. E. Leiserson, R. Rivest, C. Stein, Introduction to Algorithms, second ed., The MIT Press, 2001.

[3] E. Demaine, S. Mozes, B. Rossman, O. Weimann, An optimal decomposition algorithm for tree edit distance, in: Proceedings of the 34th International Colloquium on Automata, Languages and Programming, LNCS, vol. 4596, Springer, 2007, pp. 146–157.

[4] Y. Dinitz, A. Itai, M. Rodeh, On an algorithm of Zemlyachenko for subtree isomorphism, Information Processing Letters 70 (1999) 141–146.

[5] R. G. Downey, M. R. Fellows, M. R., Parameterized Complexity, Springer, 1999.

[6] J. Flum, M. Grohe, Parameterized Complexity, Springer, 2006.

[7] D. Fukagawa, T. Akutsu, Fast algorithms for comparison of similar unordered trees, International Journal of Foundations of Computer Science 17 (2006) 703–729.

[8] M. M. Halldórsson, K. Tanaka, Approximation and special cases of common subtrees and editing distance, in: Proceeding of the 7th International Symposium on Algorithms and Computation, LNCS, vol. 1178, Springer, 2007, pp. 75–84.

[9] D. Harel, R. E. Tarjan, Fast algorithms for finding nearest common ancestors, SIAM Journal on Computing 13 (1984) 338–355.

[10] T. Horesh, R. Mehr, R. Unger, Designing an A* algorithm for calculating edit distance between rooted-unordered trees, Journal of Computational Biology 13 (2006) 1165–1176.

[11] P. Kilpeläinen, H. Mannila, Ordered and unordered tree inclusion, SIAM Journal on Computing 24 (1995) 340–356.

[12] D. Shasha, J. T-L. Wang, K. Zhang, K., F. Y. Shih, Exact and approximate algorithms for unordered tree matching, IEEE Transactions on Systems, Man, and Cybernetics 24 (1994) 668–678.

[13] K-C. Tai, The tree-to-tree correction problem, Journal of ACM 26 (1979) 422–433.

[14] K. Zhang, R. Statman, D. Shasha, On the editing distance between unordered labeled trees, Information Processing Letters 42 (1992) 133–139.

[15] K. Zhang, T. Jiang, Some MAX SNP-hard results concerning unordered labeled trees, Information Processing Letters 49 (1994) 249–254.

[16] K. Zhang, A constrained edit distance between unordered labeled trees, Algorithmica 15 (1996) 205–222.